S19-63
80
188101

Real-Time Hierarchically Distributed Processing Network Interaction Simulation

JJ57 4450

W.F. Zimmerman and C. Wu
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

1. Abstract

The Telerobot Testbed is a hierarchically distributed processing system which is linked together through a standard, commercial Ethernet. Standard Ethernet systems are primarily designed to manage non-real-time information transfer. Therefore, collisions on the net (i.e., two or more sources attempting to send data at the same time) are managed by randomly rescheduling one of the sources to retransmit at a later time interval. Although acceptable for transmitting noncritical data such as mail, this particular feature is unacceptable for real-time hierarchical command and control systems such as the Telerobot. Data transfer and scheduling simulations, such as token ring, offer solutions to collision management, but do not appropriately characterize real-time data transfer/interactions for robotic systems. Therefore, models like these do not provide a viable simulation environment for understanding real-time network loading. A real-time network loading model is being developed which allows processor-to-processor interactions to be simulated, collisions (and respective probabilities) to be logged, collision-prone areas to be identified, and network control variable adjustments to be reentered as a means of examining and reducing collision-prone regimes that occur in the process of simulating a complete task sequence. The phase-one development results are presented in this paper. Results include 1) the theoretical foundation for the network flow model, 2) an overview of the simulation design and constraints, and 3) the software design. Ultimately, the simulation will be used to examine potential loading problems as out-year demo performance improvements cause increased data traffic. The simulation will also provide a systematic means of managing resulting loading problems.

2. Introduction

Distributed processing systems are becoming extremely common for passing mail between processors that are collocated in the same facility or separated by large geographic distances. Therefore, viable commercial systems have been developed that place processors on communication networks. For purposes of passing mail between processors, studies on Ethernet local network efficiencies have shown mean response frequencies on the order of 39.5 ms, with 100% of all traffic arriving by 200 ms (Ref. 1). Average utilization (on the order of 122 bytes per packet) with 10 hosts (processors) on the net indicates mean arrival times on the order of 10 ms (Ref. 1). Under normal operating conditions, the above response times are excellent. If variable hosts require an intermediate protocol package to ensure message consistency, then an additional mean overhead of 7 to 10 ms is not unusual. Again, if mail passing is the primary occupation of the network, then 20-50 ms is perfectly acceptable.

The Telerobot Testbed is presently employing an Ethernet (with DECnet protocol) system to facilitate interprocessor communication within the overall system computer hierarchy. Where normal mail passing finds response times on the order of 20-50 ms perfectly acceptable, robotic systems find delays in excess of 10 ms undesirable. The key reason why it is important to minimize signal delays is potential control instabilities at the lowest level of the control hierarchy (i.e., the manipulator end-effector servo control level). In hierarchical command and control systems, commands must be passed from the system exec level through several intermediate control levels before they reach the servo control level. In the Telerobot, the operator acts as the system exec by confirming the automated task sequence with the AI planner (the next level). The planner forwards high-level task commands to the run-time controller (the third level in the hierarchy) where each task is broken down into a string of primitives containing important end-point state variables, trajectory via points, and force/torque information. The run-time controller then forwards requests and commands to the manipulator control and sensing/perception processors for control execution (the fourth level of the hierarchy). Additionally, because of mismatches between processor protocols, a network interface package is necessary to maintain protocol consistency over the net. Even though there is a discrete hierarchy for command passing, each processor will be simultaneously managing outgoing commands and incoming requests (i.e., requests for world-state updates from higher level processors). In a static environment in which all fixtures are stationary, the effect of collisions on the net (two or more hosts trying to

send data at the same time) will be to merely degrade the rate at which the manipulators and end-effectors respond to incoming commands. However, in a dynamic environment in which objects or obstacles are moving (such as a rotating satellite), the degradation in speed could cause significant control problems and potential damage to the manipulators and end-effectors.

Techniques have been developed which ensure that network collisions are minimized. One of the most popular methods is token ring. The token ring technique basically assumes that message packets arrive according to a random process (Ref. 2). A single control token circulates around the ring from one host processor to the next. When a host observes that the token has been received, a data packet is queued for transmission. The token basically completes an array of "and" gate inputs and allows the packet to be transmitted. Upon completion, the token is passed to the next host, and so on, until it returns to enable another packet to be sent. Although this approach minimizes collisions on the net, it does not enable asynchronous network traffic (such as would be experienced by the telerobot) to propagate back and forth. For example, if an emergency stop signal was required in response to a calculated error, then control problems could arise if the processor that needed to send the signal had to wait a full cycle to receive the token.

Therefore, in developing a simulation for the Telerobot command and control hierarchy, attention had to be paid to developing a more stochastic network event process. Stochastic petri-nets (Ref. 3) and stochastic activity networks (Ref. 4) provided the most fruitful basis for modeling and simulating message traffic on the telerobot distributed processing network. These models were useful because they basically model 1) the arrival of packets, 2) the queuing of packets, 3) the propagation of data, and 4) the detection of collisions. The Telerobot distributed processing hierarchy can be characterized as shown in Figure 1.

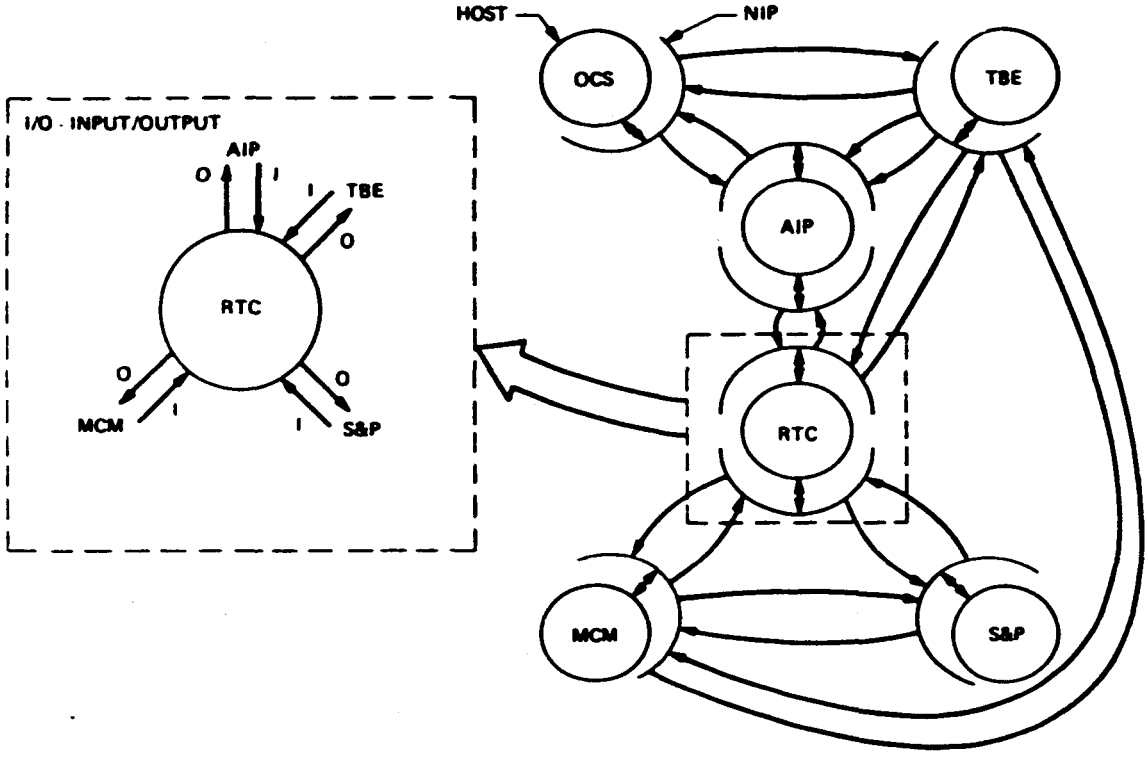

Figure 1. Telerobot Distributed Processing Hierarchy (Automated Control)

Figure 1 shows the operator control station (OCS) interfacing with the testbed exec (TBE) and the AI planner (AIP). During teleoperation the intermediate levels of the hierarchy are bypassed and the OCS/TELEOP interfaces directly with the manipulator control mechanization (MCM) and sensing/perception (S&P) subsystems. The TBE also interfaces with all other subsystems. However, for the FY 1987 and 1988 demos any robust control capability for the TBE will be suppressed; only system initialization and configuration will exist, which primarily requires a one-on-one interface with each subsystem in the hierarchy during start-up. At the AIP level of the hierarchy the planner interfaces with the OCS, TBE, and the run-time controller (RTC). Again, because of the hierarchical design, the TBE and OCS interfaces occur respectively during the system start-up and initialization phase, followed by the task sequence (menu) confirmation phase. The RTC interface occurs alternately as each task element in the sequence is presented to the operator for review/confirmation after the RTC has retrieved and forwarded the various world-state parameters to fill the menu. The RTC

interface is also initiated after the menu has been appropriately completed and the AIP has forwarded the first set of execution commands. The RTC then receives those commands and proceeds to pass specific control primitives down to the MCM and S&P subsystems. It should be noted that during task execution at the MCM and S&P level of the hierarchy (the servo control level), any of the upper levels (i.e., the AIP or RTC) can request status reports to monitor task progression and update their respective world-state data bases. The reader should note the indicated interaction between the MCM and S&P subsystems. This interaction exists during the execution of task macros (e.g., grappling a slowly rotating satellite). During the execution of a task macro, communication with the upper levels of the hierarchy can be temporarily delayed to insure that communication delays are minimized in order to prevent potential control instability associated with the dynamic environment. For the FY 1987/1988 demos, the MCM and S&P subsystems will interface with each other through either a parallel interface (in which case the net will be bypassed) or the planned DECnet network interface package (NIP). Figure 1 clearly shows that the area of concern, in terms of asynchronous requests, potential collisions, and subsequent delays in execution, is at the third level of the hierarchy (RTC). Table 1 summarizes the various operational modes, processor involvement, data transmission frequencies, and data rates as expected for the Telerobot Testbed.

Table 1. Expected Testbed Processor Performance

Operational Mode	Processors Involved	Freq. of Data Trans.	Order of Mag. Data Rate (Bits/Sec)
Start-up/Shut-down	OCS - TBE TBE - AIP TBE - RTC TBE - MCM/S&P	$<10^{-3}$ Hz	$<10^{2}$ bps
Status/Initial-ization	TBE - OCS TBE - AIP TBE - RTC TBE - MCM/S&P AIP - RTC RTC - MCM/S&P	10^{-1}-10^{-2} Hz	10^{3} bps
Planning	OCS - AIP AIP - RTC RTC - MCM RTC - S&P	.5 - 1 Hz	10^{3} bps
Execution - Autonomous	AIP - RTC RTC - MCM/S&P	RTC - 36 Hz	10^{3}-10^{4} bps
	TBE - AIP/RTC/MCM/S&P	MCM - 36 Hz	10^{3}-10^{4} bps
		S&P - 5 Hz/50 Hz during vision servoing	10^{4}-10^{5} bps
- Teleop	OCS - Teleop Teleop - MCM OCS - MCM OCS - S&P	$\leq$ 5 Hz	10^{3} bps

Considering the above expected loading, the network activity is very low for the start-up, status/initialization, and planning operational modes. This is because at any given time only two processors are actually talking to each other. This is one of the advantages of the hierarchical design. Furthermore, the communication is at a fairly low rate and on the order of "question-answer" type interactions. Since the bandwidth of the Ethernet is on the order of 10 Mbps, the network will only be utilized at a max of .001-.01% capacity. The planning mode similarly falls into the low utilization category. As illustrated by Figure 1 and confirmed by the above table, the area of concern revolves around the autonomous control execution mode. Both the transmission frequency and data rates increase substantially. Even though the utilization only increases to .1-1% capacity, the concern with this particular portion of the system in terms of modeling arises from projected substantial increases in the out-year utilization. For modeling this critical area in the distributed processor configuration it appears that at some time in the task execution mode, three to five processors might be competing for access to the net. Also, the longer the net is occupied by one processor, the greater the odds become that more than one processor will be competing for access to the net. The major overhead variables that surface from the above

design and discussion (and confirmed by the literature (Refs. 3 and 4)) are as follows:

1. The number of processors (hosts) attempting to communicate with each other at one time.
2. The frequency at which the processors communicate.
3. The size of the data blocks being communicated.
4. The internal subsystem processing time per standard data block.
5. The internal NIP protocol delay.
6. The queue time for backlogged data packets before they get transmitted (from the NIP to the host).
7. The retransmission delay resulting from a collision.
8. The Ethernet transmission interval per data block.

For this phase-one development activity, the simulation will only model network interactions involving the AIP, RTC, MCM, and S&P. The frequency at which the processors communicate or transmit data packets will be synchronized for task execution commands (i.e., a data packet sent by processor n to processor n+1 must be acknowledged on receipt before processor n+1 can send a packet to processor n+2) and randomly selected for data update requests. The size of the data blocks being transmitted will be randomly selected within the respective bps ranges given in Table 1. The NIP internal overhead will be a constant (i.e., a mean value of 7 ms). The queue time delay will be 0 if the queue is empty, and increased accordingly as the queue increases based on actual experience with the NIP. The retransmission interval will be based on the Ethernet hardware specs and each retransmission time will be selected randomly within that interval. The internal processing time will be established by multiplying the specified average hardware internal overhead per data block times the randomly selected data block size. Similarly, the Ethernet overhead is determined by multiplying the inverse of the 10 Mbps times the randomly selected data block size. Since all processors are collocated in one facility, distance will not be a factor in the Ethernet overhead.

3. Model Description

The basic problem is to establish the time interval required to move an event (e) successfully from processor (n) to processor (n+1), which in turn must move the event to processors (n+2) and (n+3). If, at any time, the state processing and sojourn times for an event generated by a given processor are equal to (or overlap) the state processing and sojourn times for an event generated by an interacting processor, then a collision occurs and the respective events receive an additional retransmission time delay, are placed in a queue, and a new state is calculated/tested to determine if a collision occurs. The event must pass successfully through the four processors before it is discarded and a new event randomly selected. Before the event is discarded, the vital statistics of 1) total sojourn time, and 2) number of collisions are collected and stored for calculating the overall system delay and collision probabilities. The events and interactions are structured as an input/output flow problem with each processor being linked by tagged events (e.g., an event that successfully leaves the AIP (processor n=1) as an output becomes an input to the RTC (processor n+1=2), and so on through the hierarchy). Mathematically, the overhead state time interval (S) for processor n and event 1 (e=1) can be stated as the sum of all internal processor state overheads (s_o):

$$S_{e_o} = \sum_{e=1}^{l} \sum_{o=1}^{p} s_{e_o} \Big|_{n=1} \tag{1}$$

where, the state overheads are the delays indicated in Section 2 above.

By not suppressing the upper bound on the event summation, the total overhead time for processor (n) can be calculated using:

$$S_{e_o} = \sum_{e=1}^{u} \sum_{o=1}^{p} s_{e_o} \Big|_{n=1} \tag{2}$$

where (u) represents the total events or commands and requests associated with a given task sequence. It then follows that the total delay (S tot) associated with events passing through the hierarchy (in this case only the AIP (n=1), RTC (n=2), MCM (n=3), and S&P (n=4) can be given as:

$$S_{tot} = \sum_{n=1}^{4} S_{e_o} \tag{3}$$

Conditionally, if an event arrival at a processor conflicts with another event input/output arrival at the same processor, then a collision (and additional delay) is imposed. The following collision conditions hold for a given event arrival:

either,

$$\sum_{o=1}^{p} s_{e_o} \Big|_{n=1} - \sum_{o=1}^{p} s_{e_o} \Big|_{n=n+1} \tag{4}$$

or, given the Ethernet overhead (E)

$$\sum_{o=1}^{p} (s_{e_o}) - E_{e_o} \Big|_{n=n+1} < \sum_{o=1}^{p} s_{e_o} \Big|_{n=1} < \sum_{o=1}^{p} s_{e_o} \Big|_{n=n+1} \tag{5}$$

The above later condition simply means that the net is occupied and processor n+1 cannot accept the command from processor n. The command from processor n would then be placed in processor n+1's queue and assigned a delay interval commensurate with its place in the queue.

Calculation of the collision probability for each processor, and the hierarchy as a whole, is done by merely employing standard statistical expressions using the logged collisions and total events processed.

The last major constraint employed for the model was a definition of "real-time." This was necessary since collisions are really only important if they indeed cause a subsequent degradation in system performance (i.e., speed). Therefore, after each event ripples through the hierarchy and the collision/time delay data are logged, the system delay is compared against the actual lag time of the system to determine if the lag time was exceeded. It is presently planned to use the manipulator control lag as the real-time baseline. For example, this means that if control lag for manipulator movement (under the planned operating speed) is 20 ms, then a collision aggregate delay of 15 ms would not affect operating speed at all. However, as response and operating speeds increase with improvements in manipulator control design, acceptable aggregate delays might be driven down into the 2-3 ms range.

The above model is still being examined from the standpoint of completeness and available data. As the software development proceeds and data bases are assembled, additional changes may be incorporated to further define the processor interaction environment. The next section briefly summarizes the software program being developed to implement the model.

4. Description of Simulation Program

In this section we will illustrate the construction of an event scheduling simulator for the network model discussed in the previous sections. In this simulator each event, created and scheduled with a time tag, represents the main activity of the network model. This time tag decides the ordering of events in the event pool of the system. This ordering also determines the execution of the events and therefore describes the operation of the network model. During the operation, there is always one available event.

As shown in Figure 2, the simulation program consists of six major components: initialization, event access, housekeeping, network activity, event handling, and report generation. The initialization module prompts the required system parameters such as number of desired RTC commands, initializes global variables, and sets the initial conditions of the system. The event access module uses a time tag and event number for each event to select a most recent event for execution. The execution of an event means activation of the event, transmission of the communication packet, and processing of the command actions if any. The housekeeping module advances the system clock based on the time tag of the current event, generates data requests periodically, and collects the required statistics data from the network activity module. The network activity module evaluates the interactions (state overhead) of the available events in the Ethernet network. The event handling module performs the processing of the command actions to generate further activity in the system. It contains several handlers to deal with different types of events. The report generation module assembles the required system delay and collision probabilities once the simulation terminates. The system will terminate when the event access module cannot get an available event, which means the event pool is empty.

Among these six modules, only the network activity module and event-handling module directly relate themselves to behavior of the network model. Therefore, they will be discussed in more detail.

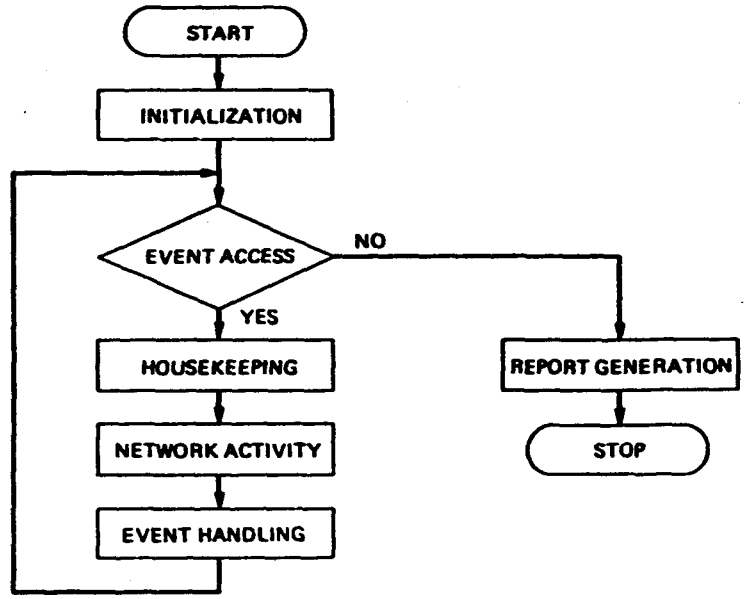

Figure 2. The Main Flow Diagram of the Simulation Program

The flowchart for the network activity module is shown in Figure 3. The main function of this module is to detect and record the collision of events. Collision occurs when two or more events are trying to occupy the network at the same time. Therefore, in order to detect the collision, the simulator computes the period of the current event residing on the network and with a given processor, and then searches through the event pool to check if any other events in the event pool will be arriving during this same period. If a collision occurs, all the involved events will be removed from the event pool and be appended to the network queues associated with the processor. Note that a random time delay will be added to each collided event for retransmission. If no collision occurs, the event becomes active and is ready to be processed (passed to the next processor) by the event handling module. The passed event is an output of the latter processor and an input to the next processor.

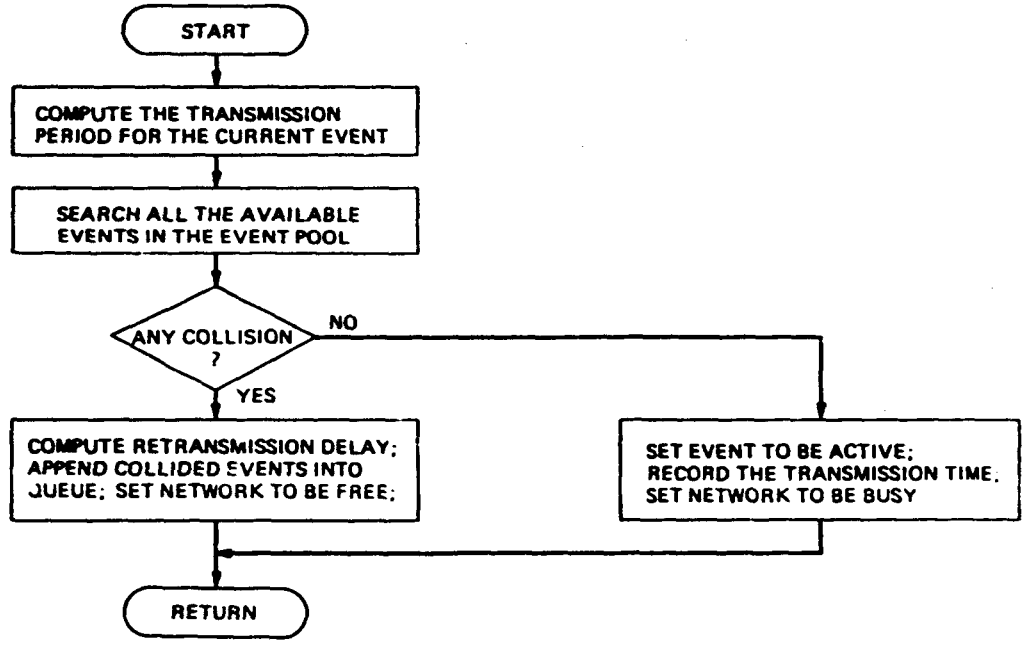

Figure 3. The Flow Diagram of the Network Activity Module

The flow diagram of the event-handling module is shown in Figure 4. There are four types of events in this simulator: command, acknowledge, request, and response. They will

be handled differently. Once a command event is received, the subsystem will generate an acknowledge event back to the originating subsystem. Then the requested command is passed to the next processor. Concurrently, the AIP and the RTC processors use request events to acquire data from lower level of the hierarchy to update their own data bases periodically. The response event is the answer for the command and request events. Since the command event is executed sequentially, the completion of the current event also activates the next command event until all events are executed in the dummy task sequence.

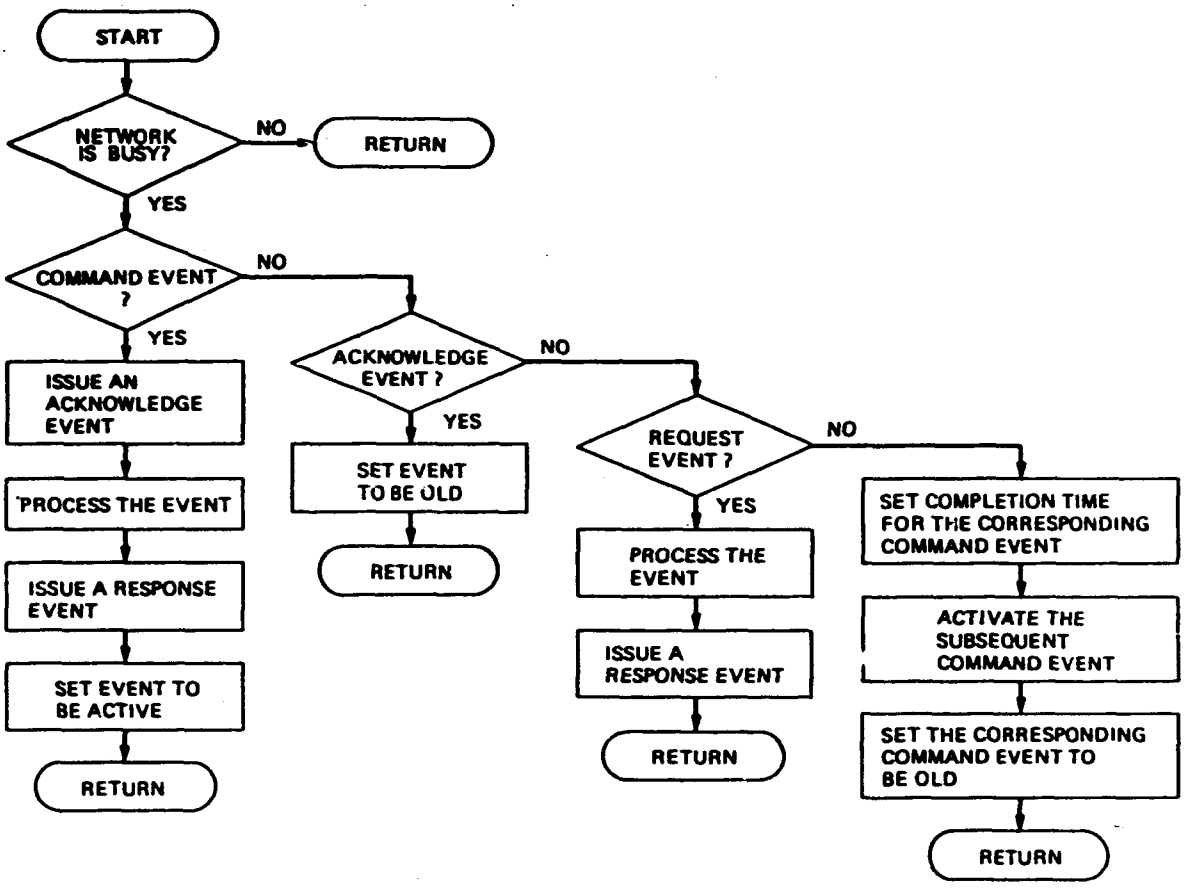

Figure 4. The Flow Diagram of the Event-Handling Module

5. Conclusions

In this paper we discuss the simulation model for a real-time hierarchically distributed system using an Ethernet local area network. The probability and the impact of message collision on the network is the main interest. The conceptual design of the simulation program was completed. The implementation is under way.

Although the simulation is not operable at this time, a considerable amount of information concerning network design and operation has been obtained. This information has been used to design the Telerobot processor protocol and communication architectures in such a manner so as to minimize network interference. These design features include 1) a high bandwidth operating environment (i.e., 10 Mbps) to minimize the overhead on the Ethernet, 2) the overall hierarchical design which prioritizes and reduces the functions that lower levels in the hierarchy must perform, 3) inclusion of a special parallel interface at the servo control level of the hierarchy to facilitate off-net high data rate transfer during critical dynamic coordination tasks (e.g., satellite grappling), 4) adjustment of the Ethernet retransmission interval to correspond with processor priority relative to control execution (e.g., during error management, the RTC and MCM processors require greater access to the net than the AIP), and 5) the inclusion of a queue I/O in the network interface protocol package to accept incoming commands/requests even though the net is occupied.

Using this simulator we can evaluate the system performance in terms of collisions during message transfer on the network, network utility, the data packet retransmission delays, and data request rate. For example, we can determine the response of the network to a utility level (i.e., transmission frequency), establish the lower and upper bound of the

data packet size, obtain an optimal data request rate, calculate new data flow control parameters to minimize collisions, and consequently resolve any system bottlenecks.

6. Acknowledgments

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the Office of Aeronautics and Space Technology (OAST), National Aeronautics and Space Administration (NASA).

7. References

[1] J.F. Shoch and J.A. Hupp, "Measured Performance of an Ethernet Local Network," Communication of the ACM, Vol. 23, No. 12, December 1980.

[2] P. Hass and G. Shedler, "Regenerative Simulation Methods for Local Area Computer Networks," IBM J. Res. Develop., Vol. 29, No. 2, March 1985.

[3] M.A. Marsan et al., "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," ACM Transactions on Computer Systems, Vol. 2, No. 2, May 1984, pp. 93-122.

[4] M.H. Woodbury and W.H. Sanders, "An Ethernet Model Simulation Using Stochastic Activity Networks," draft report, Oct. 1986.

[5] B.W. Stuck and E. Arthurs, "A Computer and Communications Network Performance Analysis Primer," Prentice Hall, 1985.

[6] H. Kobayashi, "Modeling and Analysis: An Introduction to System Performance Evaluation Methodology," Addison-Wesley, 1981.